REMARKS

Status of Claims

The status of the claims in this application is as follows:

Claims 1-25: cancelled.

Claims 26-41: rejected.

Claim 42: cancelled.

Status of Amendments

The Examiner has indicated that the amendments made after final rejection would be entered for the purposes of appeal.

That said, while claim 42 was cancelled in the response filed after final rejection, the Examiner has indicated in the Advisory Action that claims 26-42 would stand rejected for the purposes of appeal. Notwithstanding this statement, it is assumed that the cancellation of claim 42 would be entered for the purposes of appeal.

Summary of Initial Response

In response to the final Office Action, Applicant has filed a response (Amendment C) that addresses the §112 issues raised by the Examiner and, also, contends that the Examiner has not set forth a proper prima facie art-based rejection. The Examiner has indicated, in the Advisory Action, agreement that the §112 issues have been satisfactorily addressed.

However, the Examiner has also indicated, in the Advisory Action, that it is the Applicant who has not met the burden of distinguishing the claims from the cited reference. While Applicant continues to contend that there is no burden on Applicant to rebut an improperly made rejection, Applicant is submitting this supplement response (Supplemental Amendment C) in a good faith attempt to distinguish the subject matter recited in the claims from the cited reference.

Summary of Claimed Subject Matter

Before attempting to distinguish the subject matter recited in the claims from the cited reference, Applicant would like to provide a concise explanation of the subject matter defined in each rejected independent claim. In the process, Applicant refers to the specification by page and line number, and to the figures reference characters. This explanation is intended as a guide in distinguishing the subject matter recited in the claims from the cited reference, and it is not intended to itself limit the claims in any way.

Claim 26

Claim 26 is directed to a method of processing a custom action tag in a Web Page to provide a tag library extension that can be used to implement a tag library supporting multiple scripting languages. As discussed at page 12, lines 9-10 (all citations are to the specification as

filed, and not to the substitute specification), "a tag extension mechanism is a specialized sub-language that enables the addition of new or custom actions . . ."

Furthermore, at page 12, lines 14-16, it is discussed that "The tag extension mechanism of the present invention can be used from JSP pages written using any valid scripting language, although the mechanism itself only assumes a Java Run Time environment." Figure 6 illustrates an example of a tag library, including a doStart() method, a doBody() method and a doEnd() method for a custom action.

At page 14, lines 5-6, it is discussed that; "As shown in Figure 6, the code of Figure 4 can be translated using the present invention into Java code, without using a closure." Figure 4, on the other hand, illustrates how a tag can be translated (undesirably) into a closure. See page 5, line 7. That is, as discussed at page 5, lines 19, "A closure is difficult to create, however, since it is difficult to provide the requisite context. In fact, the Java programming language does not support the use of closures."

Significantly, claim 26 recites a method for processing a custom action tag where, rather than creating a complex "closure" abstraction, the evaluation of the body is "in-lined." See page 13, lines 20-21.

With that as background, we now turn to Figure 5. Figure 5 illustrates an example of the method recited in claim 26, to process a custom action tag. For example, still referring to Figure 5, the tag-handler object represents a run-time representation of a custom action tag.

In Figure 5, the do-start method is f.doStart(); the do-body method is f.doBody(); and the do-end method is f.doEnd().

Claim 26 recites, in part:

> wherein said do-start method determines:
> whether the custom action tag has a body, and
> whether there is a need to process said body when said do-start method determines that said custom action tag has a body;

Referring still to Figure 5, the f.doStart() method processes the start tag and determines if there is a body and whether there is a need to process the body (result "b").

Claim 26 further recites:

> invoking said do-body method of said tag-handler object when said invoking of said do-start method determines that there is a need to process said body of said custom action tag;
> processing, by said do-body method of said tag-handler object, said body of said custom action tag, to translate said body from a first scripting language to platform independent code that can be executed to perform actions intended by said custom action tag;
> determining by said do-body method of said tag-handler object whether further processing is required to translate the body from a first scripting language to platform independent code that can be executed to perform the actions intended by said custom action tag when said processing has been performed by said do-body method of said tag-handler object,

repeating said processing, by said do-body method of said tag-handler object, when said do-body method of said tag-handler object determines that further processing is required; and

invoking said do-end method of said tag-handler object when said do-body method determines that no further processing is required, wherein said do-end method processes said end-tag of said custom action tag.

Figure 6 illustrates both invoking the do-body method, and the processing of the do-body method, at f.doBody(). The result "b" of b=f.doBody() in Figure 6 indicates a determination of whether further processing is required, and "if (b) goto repeat" in Figure 6 corresponds to the step of repeating do-body method processing, recited in claim 26. Finally, f.doEnd() in Figure 6 corresponds to the recitation of "invoking said do-end method."

Claim 35

Claim 35 is directed to a computer readable media. The computer readable media includes computer program code for processing a custom action tag in a Web Page to provide a tag library extension that can be used to implement a tag library supporting multiple scripting languages. Given the similarities of claim 35 to claim 26, a separate concise explanation specifically directed to claim 35 is not provided. Rather, Applicant incorporates the concise explanation provided above regarding claim 26.

Claim 40

Claim 40 is directed to a computer system. The computer system includes computer program code capable of processing a custom action tag in a Web Page to provide a tag library extension that can be used to implement a tag library supporting multiple scripting languages. Given the similarities of claim 40 to claim 26, a separate concise explanation specifically directed to claim 40 is not provided. Rather, Applicant incorporates the concise explanation provided above regarding claim 26.

Grounds of Rejection

Claims 26-42 were rejected in the office action (mailed September 23, 2004) under 35 U.S.C. § 102(a) as being anticipated by Alex Federov, et al., "Professional Active Server Pages 2.0," 1998, Wrox Press Ltd (hereinafter [Federov]).

Argument

Anticipation by [Federov98]

For a reference to anticipate the subject matter of a claim, the reference must disclose each and every feature recited in that claim. Applicant respectfully contends that, as the rejection is best understood by Applicant (i.e., subject to the "Request for Clarification" made by Applicant, and rejected by the Examiner as inapplicable), [Federov98] fails to disclose each and

every feature recited in Applicant's claim 26. Thus, [Federov98] does not anticipate Applicant's claim 26. Applicant's contentions are discussed in detail below.

A. Do-Start Method

The Examiner contends that an Application_onStart event handler or a Session_onStart event handler corresponds to the do-start method recited in claim 26 as being included in a tag-hander object. The Examiner cites to [Federov] as follows:

> see, for example, "Application Event Handlers" on pp. 132-133, "Session Event Handlers" and "Session Variables" on pp. 138-139, and "Application and Session Events" on pp. 328-329)"

With regard to the feature of "invoking said do-start method . . . ," the Examiner's citations are the same, except that, for the recitation of what the do-start method determines

> whether the custom action tag has a body

and

> whether there is a need to process said body when said do-start method determines that said custom action tag has a body

the Examiner cites as follows: "see, for example, 'Applicant, Sessions and State' on pp. 325-341."

With regard to pp. 132-133 and pp. 138-139, these pages clearly do not disclose that the onStart handlers (Application_onStart and Session_onStart) make the determinations recited in claim 26. Rather, these pages disclose only "Insert script to be executed when the application starts" (for Application_on_Start) and "Insert script to be executed when a session starts" (for Session_onStart). In addition, pages 328-329 disclose that the onStart events are used to initialize state, but there is no disclosure of determining whether there is a body and whether there is a need to process the body. Specifically, this section discloses the following with respect to the Application_onStart and Session_onStart events:

> Both of these events are used to initialize state, by setting up variables that are global for the application or for a specific user. When the first user accesses a file in our application, the **Application_onStart** event is triggered. This is used to initialize any application-wide global variables. When the user begins a session for the first time, the **Session_onStart** event is triggered. This is used to initialize user-specific information:
>
> The power of the **Session** object comes from the fact that it can store variables that are global to just that specific user, and so each user can have their own individual value for that variable. **Session** objects aren't always created automatically for every user when they enter our application. However, storing or accessing a variable in the Session object will create it, and fire the **Session_onStart** event. We can force to sessions to always be created as soon as a visitor enters our application by writing code in **global.asa** to respond to this event.

We note that firing the Session_onStart event when a variable in the Session object is created, based on storing or accessing the variable, is not the same as determining whether there is a body and whether there is a need to process the body.

This leaves pp. 325-341, generically, of [Federov98] as the sole remaining possible source for disclosing the determinations recited in claim 26 as being carried out by the do_start

method (i.e., whether there is a body and whether there is a need to process the body). That is, we have specifically distinguished the disclosure at pp. 132-133, pp. 138-139, and pp. 328-329; and the Examiner cites pp. 325-341 generally as support for the contention that [Federov98] discloses the determining steps of the do-start method.

Applicant has made a good faith attempt to find a disclosure of the Application_onStart and Session_onStart handler making the determinations recited in claim 26. Besides a general disclosure that the onStart handlers are uses to initialize variables, the only other disclosure Applicant can find of processing within an onStart handler is at page 334 and 336, which disclose creating an instance of an object if one does not already exist. For example, at page 334, there is the following code section disclosed as being part of a Session_onStart routine:

```
If IsEmpty(Application("Object")) Then
    Set Application ("Object") = Server.CreateObject ("global.connection")
    Application ("ObjectCount") = 0
End If
```

At page 336, there is a similar code section:

```
If IsEmpty(Application("myData")) Then
    Application ("myData") =strTheValue
End If
```

Neither of these disclosures of onStart hander processing disclose determining whether there is a body and whether there is a need to process the body.

B. Do-Body Method

The Examiner also contends that [Federov98] discloses do-body method processing "to translate said body from a first scripting language to platform independent code that can be executed to perform actions intended by said custom tag." The Examiner specifically refers to " 'The Response Object' on pp. 113-125 and 'The Server Object' on pp. 125-131"). Applicant can find no disclosure of this recited do-body method processing in the cited portions of [Federov98].

In particular, there is no disclosure in the cited portions of [Federov98] (as best understood by Applicant, without the benefit of a more specific contention by the Examiner, requested and refused) of a "body" of a custom action tag (i.e., wherein "said custom action tag further capable of having a body between said start-tag and said end-tag") being translated from a first scripting language to platform independent code.

In fact, the Examiner has not even pointed out where the cited portions of [Federov98] are contended to disclose a "body" (again, between a start-tag and an end-tag) at all, and Applicant respectfully contends that the cited portions of [Federov98] do not include such a disclosure.

Furthermore, there is no disclosure in the cited portions of [Federov98] (again, as best understood by Applicant) of "determining by said do-body method of said tag-handler object

whether further processing is required to translate the body from a first scripting language to platform independent code . . ."

C. Claims 35 and 40

Applicant has specifically pointed out several features recited in claim 26 that are not present in the cited portions of [Federov98]. With respect to claims 35 and 40, similar features are recited in those claims. Thus, in distinguishing the recitations of claims 35 and 40 from the cited portions of [Federov98], Applicant incorporates the contentions made above to distinguish the features of claim 26 from the cited portions of [Federov98]

CONCLUSION

It is once again respectfully submitted that the Examiner has failed to comply with 37 CFR 1.104(c)(2), and Applicant respectfully renews the request for a more specific designation of the portions of [Federov98] relied upon in rejecting the claims.

In addition, it is respectfully submitted, as [Federov98] is understood by Applicant without the benefit of such specific designations, that [Federov98] does not disclose each and every feature recited in the rejected claims. Applicant has specifically pointed out, using the language of the claims, how the cited portions of [Federov98] do not disclose particular features recited in the claims.

Applicant respectfully requests that the anticipation rejection over [Federov98] be withdrawn and that the application be passed to allowance.

Respectfully submitted,
BEYER WEAVER & THOMAS, LLP

Alan S. Hodes
Reg. No. 38,185

P.O. Box 778
Berkeley, CA 94704-0778
(650) 961-8300